Evolução do HTTP

O HTTP (HyperText Transfer Protocol) é o protocolo subjacente da World Wide Web. Desenvolvido por Tim Berners-Lee e sua equipe entre 1989-1991, o HTTP passou por muitas mudanças que ajudaram a manter sua simplicidade ao mesmo tempo em que moldaram sua flexibilidade. Continue lendo para aprender como o HTTP evoluiu de um protocolo projetado para trocar arquivos em um ambiente de laboratório semiconfiável até se tornar um labirinto moderno da internet que carrega imagens e vídeos em alta resolução e 3D.

Invenção da World Wide Web

Em 1989, enquanto trabalhava no CERN, Tim Berners-Lee escreveu uma proposta para construir um sistema de hipertexto sobre a internet. Inicialmente chamado de *Mesh*, mais tarde foi renomeado como *World Wide Web* durante sua implementação em 1990. Construído sobre os protocolos existentes TCP e IP, consistia em 4 blocos fundamentais:

- Um formato textual para representar documentos de hipertexto, a **HyperText Markup Language (HTML)**.
- Um protocolo para trocar esses documentos, o **HyperText Transfer Protocol (HTTP)**.
- Um cliente para exibir (e editar) esses documentos, o primeiro navegador chamado **WorldWideWeb**.
- Um servidor para dar acesso ao documento, uma versão inicial do **httpd**.

Esses quatro blocos foram concluídos até o final de 1990, e os primeiros servidores estavam operando fora do CERN no início de 1991. Em 6 de agosto de 1991, Tim Berners-Lee publicou uma mensagem no grupo de notícias público *alt.hypertext*. Este é agora considerado o início oficial da World Wide Web como um projeto público.

O protocolo HTTP usado nessas fases iniciais era muito simples. Posteriormente foi chamado de **HTTP/0.9** e às vezes é referido como o protocolo de uma linha só.

HTTP/0.9 – O protocolo de uma linha

A versão inicial do HTTP não possuía número de versão; mais tarde foi chamada de 0.9 para diferenciá-la de versões posteriores. O HTTP/0.9 era extremamente simples: as requisições consistiam em uma única linha e começavam com o único método possível, **GET**, seguido pelo caminho para o recurso. A URL completa não era incluída, pois o protocolo, servidor e porta não eram necessários uma vez conectados ao servidor.

A resposta também era extremamente simples: consistia apenas no próprio arquivo.

Diferente das evoluções subsequentes, não havia cabeçalhos HTTP. Isso significava que apenas arquivos HTML podiam ser transmitidos. Não havia códigos de status ou erro. Se houvesse um problema, um arquivo HTML específico era gerado e incluía uma descrição do problema para compreensão humana.

HTTP/1.0 – Construindo a extensibilidade

O HTTP/0.9 era muito limitado, mas navegadores e servidores rapidamente o tornaram mais versátil:

- Informações de **versão** passaram a ser enviadas em cada requisição (HTTP/1.0 era adicionado à linha GET).
- Uma **linha de código de status** também passou a ser enviada no início da resposta. Isso permitia que o navegador reconhecesse o sucesso ou falha da requisição e adaptasse seu comportamento, por exemplo, atualizando ou usando seu cache local de forma específica.
- O conceito de **cabeçalhos HTTP** foi introduzido tanto para requisições quanto para respostas. Metadados podiam ser transmitidos e o protocolo se tornava extremamente flexível e extensível.

Exemplo:

```pgsql
HTTP
GET /my-page.html HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

HTTP/1.0 200 OK
Date: Tue, 15 Nov 1994 08:12:31 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/html

<HTML>
A page with an image
<IMG SRC="/my-image.gif">
</HTML>
```

```pgsql
HTTP
GET /my-image.gif HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

HTTP/1.0 200 OK
Date: Tue, 15 Nov 1994 08:12:32 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: image/gif

(conteúdo da imagem)
```

HTTP/1.1 – O protocolo padronizado

Enquanto isso, a padronização adequada estava em andamento, paralela às diversas implementações do HTTP/1.0. A primeira versão padronizada do HTTP, **HTTP/1.1**, foi publicada no início de 1997, apenas alguns meses após o HTTP/1.0.

O HTTP/1.1 esclareceu ambiguidades e introduziu diversas melhorias:

- **Conexão reutilizável**: não era mais necessário abrir conexões múltiplas para exibir os recursos incorporados no documento original.
- **Pipelining**: permitia que uma segunda requisição fosse enviada antes da resposta da primeira ser completamente transmitida.
- **Respostas segmentadas (chunked)** também foram suportadas.
- **Mecanismos adicionais de controle de cache** foram introduzidos.
- **Negociação de conteúdo** (idioma, codificação, tipo) permitia que cliente e servidor acordassem sobre qual conteúdo trocar.
- O cabeçalho **Host** permitia hospedar diferentes domínios em um mesmo endereço IP.

```
HTTP

GET /en-US/docs/Glossary/CORS-safelisted_request_header HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/en-US/docs/Glossary/CORS-safelisted_request_header

HTTP/1.1 200 OK
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 20 Jul 2016 10:55:30 GMT
Etag: "547fa7e369ef56031dd3bff2ace9fc0832eb251a"
Keep-Alive: timeout=5, max=1000
Last-Modified: Tue, 19 Jul 2016 00:59:33 GMT
Server: Apache
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding

(content)

GET /static/img/header-background.png HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/en-US/docs/Glossary/CORS-safelisted_request_header

HTTP/1.1 200 OK
Age: 9578461
Cache-Control: public, max-age=315360000
Connection: keep-alive
Content-Length: 3077
Content-Type: image/png
Date: Thu, 31 Mar 2016 13:34:46 GMT
Last-Modified: Wed, 21 Oct 2015 18:27:50 GMT
Server: Apache

(image content of 3077 bytes)
```

Mais de duas décadas de desenvolvimento

A extensibilidade do HTTP tornou fácil a criação de novos cabeçalhos e métodos. Apesar de o protocolo **HTTP/1.1** ter sido refinado em duas revisões — **RFC 2616**, publicado em junho de 1999, e os RFCs **7230 a 7235**, publicados em junho de 2014, antes do lançamento do HTTP/2 — ele foi extremamente estável por mais de 15 anos.

O **HTTP/1.1** foi novamente atualizado em 2022 com o **RFC 9110**. Não apenas o HTTP/1.1 foi atualizado, mas todo o protocolo HTTP foi revisado e agora está dividido nos seguintes documentos:

- **Semântica (RFC 9110)**
- **Cache (RFC 9111)** – aplicável a todas as versões do HTTP
- **HTTP/1.1 (RFC 9112)**
- **HTTP/2 (RFC 9113)**
- **HTTP/3 (RFC 9114)**

Utilizando HTTP para transmissões seguras

A maior mudança no HTTP ocorreu no final de 1994. Em vez de enviar HTTP sobre uma pilha básica de TCP/IP, a empresa de serviços de computação **Netscape Communications** criou uma camada adicional de transmissão criptografada sobre ela: o **SSL**. O SSL 1.0 nunca foi lançado ao público, mas o **SSL 2.0** e seu sucessor **SSL 3.0**

permitiram a criação de websites de comércio eletrônico. Para isso, eles criptografavam e garantiam a autenticidade das mensagens trocadas entre servidor e cliente. O SSL eventualmente foi padronizado e tornou-se o **TLS**.

Durante o mesmo período, tornou-se claro que era necessário uma camada de transporte criptografada. A web já não era mais uma rede majoritariamente acadêmica, e passou a se tornar uma selva onde anunciantes, indivíduos aleatórios e criminosos competiam por tantos dados privados quanto possível. À medida que as aplicações construídas sobre HTTP se tornavam mais poderosas e exigiam acesso a informações privadas como agendas de contatos, e-mails e localização do usuário, o **TLS** tornou-se necessário além do uso em e-commerce.

Usando HTTP para aplicações complexas

Tim Berners-Lee originalmente não via o HTTP como um meio apenas de leitura. Ele queria criar uma web onde as pessoas pudessem adicionar e mover documentos remotamente — uma espécie de sistema de arquivos distribuído. Por volta de 1996, o HTTP foi estendido para permitir autoria, e um padrão chamado **WebDAV** foi criado. Ele evoluiu para incluir aplicações específicas como **CardDAV** (para gerenciar entradas de agendas de contatos) e **CalDAV** (para lidar com calendários). Mas todas essas extensões *DAV tinham uma falha: só eram utilizáveis quando implementadas pelos servidores.

Em 2000, um novo padrão para uso do HTTP foi projetado: **REST** (*Representational State Transfer*). A API não se baseava em novos métodos HTTP, mas sim no acesso a URIs específicas com os métodos básicos do HTTP/1.1. Isso permitia que qualquer aplicação web disponibilizasse uma API para recuperar e modificar seus dados sem ter que atualizar navegadores ou servidores. Toda a informação necessária era incorporada nos arquivos que os sites serviam por HTTP/1.1 padrão.

A desvantagem do modelo REST era que cada site definia sua própria API RESTful não padronizada e tinha controle total sobre ela. Isso diferia das extensões *DAV, nas quais clientes e servidores eram interoperáveis. As APIs RESTful se tornaram muito comuns na década de 2010.

Desde 2005, mais APIs passaram a ficar disponíveis para páginas web. Várias dessas APIs criam extensões ao protocolo HTTP para finalidades específicas:

- **Server-sent events**, onde o servidor pode enviar mensagens ocasionais ao navegador.
- **WebSocket**, um novo protocolo que pode ser estabelecido atualizando uma conexão HTTP existente.

Relaxando o modelo de segurança da web

O HTTP é independente do modelo de segurança da web, conhecido como **política de mesma origem (same-origin policy)**. Na verdade, o modelo atual de segurança da web foi desenvolvido após a criação do HTTP!

Com o passar dos anos, provou-se útil suspender algumas restrições dessa política sob certas condições. O servidor indicava ao cliente o quanto e quando suspender tais restrições, usando um novo conjunto de cabeçalhos HTTP. Estes foram definidos em especificações como:

- **CORS (Cross-Origin Resource Sharing)**
- **CSP (Content Security Policy)**

Além dessas extensões principais, muitos outros cabeçalhos foram adicionados, às vezes apenas experimentalmente. Cabeçalhos notáveis incluem:

- **DNT (Do Not Track)** para controle de privacidade,
- **X-Frame-Options**,
- **Upgrade-Insecure-Requests**, entre outros.

HTTP/2 – Um protocolo para maior desempenho

Com o passar do tempo, as páginas web se tornaram mais complexas. Algumas passaram a ser verdadeiras aplicações. Mais mídias visuais passaram a ser exibidas e o volume e tamanho dos scripts adicionando interatividade também aumentaram. Muito mais dados eram transmitidos por significativamente mais requisições HTTP, o que gerava mais complexidade e sobrecarga para conexões HTTP/1.1.

Para lidar com isso, o Google implementou um protocolo experimental chamado **SPDY** no início dos anos 2010. Esse modo alternativo de troca de dados entre cliente e servidor atraiu o interesse de desenvolvedores de navegadores e servidores. O SPDY aumentava a responsividade e resolvia o problema de transmissão de dados duplicados, servindo como base para o protocolo **HTTP/2**.

HTTP/2 difere do HTTP/1.1 de várias maneiras:

- É um protocolo **binário**, não de texto. Ele não pode ser lido ou criado manualmente. Apesar disso, permite a implementação de técnicas de otimização aprimoradas.
- É um protocolo **multiplexado**. Requisições paralelas podem ser feitas na mesma conexão, eliminando as limitações do HTTP/1.x.
- Ele **comprime os cabeçalhos**. Como esses são frequentemente semelhantes entre um conjunto de requisições, isso remove duplicação e sobrecarga de dados transmitidos.

Oficialmente padronizado em maio de 2015, o uso do HTTP/2 atingiu seu pico em janeiro de 2022, com **46,9% de todos os websites**. Sites de alto tráfego mostraram a adoção mais rápida, visando economias de largura de banda e orçamento.

Essa rápida adoção se deveu, em grande parte, ao fato de o HTTP/2 não exigir mudanças em sites e aplicações. Para utilizá-lo, era necessário apenas um servidor atualizado que se comunicasse com um navegador moderno. Um conjunto limitado de grupos era suficiente para iniciar a adoção e, à medida que versões antigas de navegadores e servidores eram substituídas, o uso aumentava naturalmente, sem muito esforço por parte dos desenvolvedores web.

Evolução pós-HTTP/2

A extensibilidade do HTTP continua sendo usada para adicionar novos recursos. Entre os principais recursos surgidos em 2016 estão:

- Suporte ao cabeçalho **Alt-Svc**, que permitiu dissociar a identificação e localização de um recurso. Isso significou um mecanismo de cache mais inteligente em CDNs.
- Introdução dos **client hints**, que permitiram que o navegador ou cliente informasse proativamente ao servidor sobre seus requisitos e limitações de hardware.
- Introdução de **prefixos relacionados à segurança** no cabeçalho **Cookie**, ajudando a garantir que cookies seguros não fossem alterados.

HTTP/3 – HTTP sobre QUIC

A próxima grande versão do HTTP, o **HTTP/3**, tem a mesma semântica das versões anteriores, mas utiliza o **QUIC** em vez do TCP na camada de transporte. Em outubro de 2022, **26% de todos os websites** estavam usando o HTTP/3.

O **QUIC** foi projetado para fornecer latência muito mais baixa em conexões HTTP. Assim como o HTTP/2, ele é multiplexado. No entanto, o HTTP/2 opera sobre uma única conexão TCP, então a detecção e retransmissão de perda de pacotes feita na camada TCP pode bloquear todos os fluxos. O **QUIC**, por sua vez, executa vários fluxos sobre o

UDP e implementa detecção e retransmissão de pacotes de forma independente para cada fluxo, de modo que se ocorrer um erro, apenas o fluxo correspondente ao pacote é bloqueado.

Definido no **RFC 9114**, o HTTP/3 é suportado pela maioria dos navegadores principais, incluindo **Chromium** (e suas variantes como Chrome e Edge) e **Firefox**.